Assignment 5

Derek Paulsen

1 Problem 1

1.1 Part A

Let S be the optimal solution to the knapsack instance, consider the value of this solution,

$$value(S) = \sum_{i \in S} v_i$$

If we round the values down we get a new value of

$$value(S') = \sum_{i \in S} \left\lfloor \frac{v_i}{V\epsilon/n} \right\rfloor V\epsilon/n$$

by rounding, we decrease the value of each object by at most $V\epsilon/n$, hence we get,

$$value(S') \geq \sum_{i \in S} v_i - (V\epsilon/n)$$

We then show that

$$\begin{aligned}
\sum_{i \in S} v_i - (V\epsilon/n) &\geq (1-\epsilon)\sum_{i \in S} v_i \\
-|S|(V\epsilon/n) + \sum_{i \in S} v_i &\geq (1-\epsilon)\sum_{i \in S} v_i \\
-|S|(V\epsilon/n) &\geq -\epsilon \sum_{i \in S} v_i \\
|S|(V/n) &\leq \sum_{i \in S} v_i
\end{aligned}$$

We now note that $|S|/n \leq 1$,

$$V \leq \sum_{i \in S} v_i$$

From the setup of the problem, we have that V is the maximum value object which has weight at most W, hence any optimal solution must have value at least V since we could easily obtain a solution of value V by taking said item. This means that indeed, $V \leq \sum_{i \in S} v_i$ which implies, rounding the solution decreased the value by at most $(1-\epsilon)$.

1.2 Part B

Let $X = \{(v_i, w_i) | i \in [1, n]\}$ the set of possible items to take. We first round the value of each item as described above for some $\epsilon \in (0, 1)$. Assume we obtain $v'_1, ..., v'_k$ distinct values, where $\forall i < j, v'_i < v'_j$. Finally, let X_i be the set of all items with value v'_i around rounding. Our algorithm is then,

- $A \leftarrow$ an $k \times n$ matrix keeping track of the items used, the total value and weight
- For $i \in [1, k]$

- $y \leftarrow argmin_{x \in X_i}((weight(x))$
- $Y \leftarrow X_i \setminus \{y\}$
- For $j \in [1, n]$
  - $b \leftarrow$ the maximum value set of items that we have already computed in A which has total weight less than $W - weight(y)$ and at most $i - 1$ items
  - $c \leftarrow$ the maximum value set of items that we have already computed in A and at most i items
  - If $value(b) + value(y) = value(c)$ then $A_{ij} \leftarrow$ the combination with the least weight
  - If $value(b) + value(y) > value(c)$ then $A_{ij} \leftarrow b \cup \{y\}$ and $y \leftarrow argmin_{y \in Y}((weight(y))$ and $Y \leftarrow Y \setminus \{y\}$
  - If $value(b) + value(y) < value(c)$ then $A_{ij} \leftarrow c$

The idea of this algorithm is that at we compute the best set of at most i items only using items which have value at most v_i.

2 Problem 2

2.1 Part A

We solve this problem as follows. Suppose for $i \in [1, k]$ we have $c_i > 0$ items of size s_i. Let $x \in \mathbb{Z}_+^k$ where $x_i =$ the number of items of size s_i in the packing. We then define $f(x)$ to be the optimal packing for items specified by x.

- For $i \in [1, n]$
  - For $x \in \{x | \sum_{j=1}^{k} x_j = i \wedge x_j \leq c_j\}$ // all combinations of i items
    - Compute $f(x)$ by taking at pair of previously compute optimal packings and merging them together

We note that we compute the optimal packing for all combinations of x. There are $(\Pi_{i=1}^{k}(s_i + 1)) - 1 = O(n^k)$ different combinations of sizes to compute. Computing each packing requires examining at most $O(n^k)$ different combinations and merging previous solutions takes $O(n)$ hence the runtime of the algorithm is $O(n^{2k+1})$ which is polynomial.

2.2 Part B

Suppose we have an empty bin which we pack by sorting the elements into decreasing order of size and then adding the elements until we cannot fit the next one. Additionally, assume that the first element is of size l. We note that the amount of unused space in the bin is less than or equal to l since all of the elements in the stream we were trying to pack were of size less than or equal to l. Now suppose we have packed all the items of size $> \epsilon$ into B bins, leaving only items of size $\leq \epsilon$ left. We then proceed to three steps. First, we sort the items into order of decreasing size. Next, for each of the existing B bins, we greedily items from the sorted list until the next item doesn't fit, at which point we move to the next bin and repeat the process. Finally, we pack the remaining items into bins in the same fasion until we have exhausted all of the bins. We now consider how many bins we have used by the end of the procedure.
Let $A = \sum_{i=1}^{n} a_i$. After the second step of the algorithm above, we have two cases. In the first case, we have exhausted all of the items we needed to pack in which case we have packed all of the items into B bins. In the second case, we havn't exhausted the items, we now consider this case. From the lemma above, after out second step each of the B bins has at most ϵ wasted space, hence each bin has at least $(1 - \epsilon)$ stuff packed into it. This means that we have at most $A - \frac{B}{1-\epsilon}$ stuff left to pack into bins, again apply the lemma above we know that we can pack all of these items into at most,

$$\left\lceil \frac{A - \frac{B}{1-\epsilon}}{1 - \epsilon} \right\rceil$$

bins. Additionally by our assumption we have that $A - \frac{B}{1-\epsilon} > 0$, we now want to show,

$$\left\lceil \frac{A - \frac{B}{1-\epsilon}}{1-\epsilon} \right\rceil \leq 1 + (1+2\epsilon)B^*$$

Remove the ceil,

$$\begin{aligned}
B + 1 + \frac{A - \frac{B}{1-\epsilon}}{1-\epsilon} &\leq 1 + (1+2\epsilon)B^* \\
B + \frac{A - \frac{B}{1-\epsilon}}{1-\epsilon} &\leq (1+2\epsilon)B^* \\
B(1-\epsilon) + A - \frac{B}{1-\epsilon} &\leq (1-\epsilon)(1+2\epsilon)B^* \\
A - B\frac{\epsilon(2-\epsilon)}{1-\epsilon} &\leq (1-\epsilon)(1+2\epsilon)B^*
\end{aligned}$$

$A \leq B^*$,

$$\begin{aligned}
B^* - B\frac{\epsilon(2-\epsilon)}{1-\epsilon} &\leq (1-\epsilon)(1+2\epsilon)B^* \\
B^* - B\frac{\epsilon(2-\epsilon)}{1-\epsilon} &\leq (1+\epsilon-2\epsilon^2)B^* \\
-B\frac{\epsilon(2-\epsilon)}{1-\epsilon} &\leq (\epsilon-2\epsilon^2)B^* \\
-B\frac{\epsilon(2-\epsilon)}{1-\epsilon} &\leq \epsilon(1-2\epsilon)B^* \\
-B\frac{(2-\epsilon)}{1-\epsilon} &\leq (1-2\epsilon)B^* \\
-B\frac{(2-\epsilon)}{1-\epsilon} &\leq (1-2\epsilon)B^* \\
-B(2-\epsilon) &\leq (1-\epsilon)(1-2\epsilon)B^*
\end{aligned}$$

We now note that the right hand side of the equation is strictly positive while the left hand side is strictly negative, hence the inequality holds. Hence in both cases we use $\max\{B, 1+(1+2\epsilon)B^*\}$ bins to pack all of the items. Finally, the sorting of the items takes $O(n \log n)$ and the inserting them takes $O(n)$ hence the procedure takes $O(n \log n)$ time which is polynomial in n.

2.3 Part C

Consider the round instances, $\forall i < j, x \in S_i, y \in S_j, original\_size(x) \geq rounded\_size(y)$. Now consider the original packing, we could set aside all of the items in S_1 and then place each item from S_2 in the spot used for S_1. We could then repeat this process for each set, place the items from the set in spaces used by the previous set in the original packing and still end up with a valid packing for the items in $S_2, ...S_{n/k}$. We are then left with n/k items in S_1 which trivially fit into n/k bins (one bin per item) hence we used the original packing + at most n/k extra bins for the rounded instance.

2.4 Part D

Let S be the set of items of size greater than $\epsilon/2$. Let B be the optimal packing for S. From the question above we have that after rounding and packing the large items we have used at most $B + 1/2|S|\epsilon^2$ for $k = 1/2\epsilon^2$. From part two we get that we can pack the remaining items into $\max\{B+1/2|S|\epsilon^2, 1+(1+\epsilon)B^*\}$ bins. We now show that,

$$B + 1/2|S|\epsilon^2 \leq 1 + (1+\epsilon)B^*$$

$B \leq B^*$,

$$\begin{aligned} B^* + 1/2|S|\epsilon^2 &\leq 1 + (1+\epsilon)B^* \\ 1/2|S|\epsilon^2 &\leq 1 + \epsilon B^* \\ 1/2|S|\epsilon &\leq B^* \end{aligned}$$

$\sum a_i \leq B^*$,

$$\begin{aligned} 1/2|S|\epsilon &\leq \sum a_i \\ 1/2|S|\epsilon &\leq \sum_{i \in S} a_i \end{aligned}$$

We now note that this must be true since all items in S are at least size $\epsilon/2$. Hence $B + 1/2|S|\epsilon^2 \leq 1 + (1+\epsilon)B^*$. Which implies that we can pack the remaining items and end up with, $1 + (1+\epsilon)B^*$ bins.
Finally, from part one we show that we can pack the items the large items into bins in $O(n^{2k+1}$ time and in $O(n)$ time for the small bins, hence the algorithm runs in polynomial time.

3 Problem 3

3.1 Part A

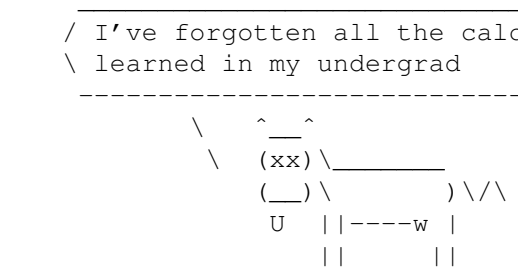
Consider a sequence of prices where the price is B until time T at which point all subsequent prices becomes 0. Clearly after time T the algorithm should buy the computer (no one is going to pay you to take a new computer right?). This of course is the exact same problem as the ski rental problem. You pay one dollar per minute (day in the ski rental) and at any point you can purchase the computer for B dollars (equivalent to buying the skis) and stop. After time T, any algorithm will stop which is equivalent to quitting skiing.

3.2 Part B

My calculus is very bad, however the idea behind that if you use the skiing problem and have set the probability of the price dropping to zero (stopping skiing) to an exponential distribution, any deterministic stopping rule will have expected competitive ratio of $e/(e-1)$. Since no deterministic algorithm can do better than $e/(e-1)$ no randomized algorithm can do better than $e/(e-1)$, hence no algorithm can do better than $e/(e-1)$.

3.3 Part C

Here is a cow.



4 Problem 4

4.1 Part A

As in lecture we wish to show that for each step,

$$\Delta ALG(\sigma) + \Delta\phi \leq k\Delta OPT(\sigma) \tag{1}$$

again we break each request into two stages, in stage one OPT moves, and stage two DC moves. Now consider the stage 1, where OPT moves distance d to service the request, consider the changes of each

- $\Delta OPT = d$, since OPT moved d
- $\Delta DC = 0$, since we didn't move yet
- $\Delta\phi_1 \leq kd$, since in the worse case, the cost of $M(x, y)$ increased by d
- $\Delta\phi_2 = 0$, since we didn't move yet

summing these terms together we get

$$0 + kd + 0 \leq kd$$

hence for stage one the inequality holds.
We now consider stage two. Let d_i be the distance that server i moves. WLOG we assume that $\{d_1, ..., d_k\}$ is sorted in increasing order, that is, server k moves the furthest and actually goes to the request, server 1 moves the least distance. For ease of exposition, we break this movement into k substeps, at step i, all servers $>= i$ move $(d_i - d_{i-1})$, where $d_0 = 0$. Now consider the pairwise distances. For when a single server moves it moves $(d_i - d_{i-1})$ units closer to the servers it doesn't cover, while moving away from the servers that it covers. We note that each server is only every covered by a single server that is moving hence at each substep, the pairwise distances increase by at most $i(d_i - d_{i-1})$, hence we get,

$(d_i - d_{i-1}) =$ the distance covered by the servers
$(k - i - 1) =$ the number of servers that are moving
$(k - 1) =$ the number of servers that the moving server is moving towards
$-2i =$ compensation for the servers that are currently covered, times 2 because the previous term counts the distance as de

We know consider each component,

- $\Delta OPT = 0$, since OPT didn't move
- $\Delta DC = \sum_{i=1}^{k} d_i$, the sum of the distances that we moved,
- $\Delta\phi_1 \leq k(\sum_{i=1}^{k-1} d_i - d_k)$, since in the worse case, all servers except for server k moved away from its matched counterpart
- $\Delta\phi_2 = \sum_{i=1}^{k}((k - i + 1)(k - 1) - 2i)(d_i - d_{i-1})$

we then wish to show,

$$\sum_{i=1}^{k} d_i + k(\sum_{i=1}^{k-1} d_i - d_k) - \sum_{i=1}^{k}((k - i + 1)(k - 1) - 2(i - 1))(d_i - d_{i-1}) \leq 0k$$

$$\sum_{i=1}^{k} d_i + k(\sum_{i=1}^{k-1} d_i - d_k) - \sum_{i=1}^{k}(k^2 - ki - i - 3)(d_i - d_{i-1}) \leq 0$$

$$\sum_{i=1}^{k} d_i + k(\sum_{i=1}^{k-1} d_i - d_k) - (k^2 - 3)d_k + \sum_{i=1}^{k}(-ki - i)(d_i - d_{i-1}) \leq 0$$

$$\sum_{i=1}^{k} d_i + k(\sum_{i=1}^{k-1} d_i - d_k) - (k^2 - 3)d_k - (k + 1)\sum_{i=1}^{k} d_i \leq 0$$

$$k(\sum_{i=1}^{k-1} d_i - d_k) - (k^2 - 3)d_k - k\sum_{i=1}^{k} d_i \leq 0$$

$$-2kd_k - (k^2 - 3)d_k \leq 0$$

$$True$$

Hence for stage two the inequality hold for stage two. This implies that

$$\Delta ALG(\sigma) + \Delta\phi \leq k\Delta OPT(\sigma) \tag{2}$$

For all steps, which implies that the algorithm is k-competitive.

4.2 Part B

Consider a star graph, where each edge corresponds to a single page and all edges are length 1. Suppose we start with all the servers in the center point of the graph and all requests are at the end points of the edges (away from the center). Suppose we apply algorithm from above, when a request comes in at the end of one of the edges of the graph, all servers technically cover each other, hence only a single server is sent to service the request. This page is then 'marked' because the server won't leave move from this point until k unique requests have come in. For subsequent requests there are three scenarios.

1. The request is where a server currently, in which case no server moves, a cache hit.
2. The request is not where a server currently is, and there is at least one server in the middle of the graph, at which point a random server will choosen to service the request, marking the page
3. The request is not where a server currently is, and there arn't any servers in the middle of the graph, at which point all servers will move to the middle of the graph 'unmarking' all of the pages (and evicting them) and then moving a random server to service the request, marking the new page.

We can see that this algorithm evicts pages unnecessarily, hence any Marking algorithm is at least as good as this algorithm. Since this algorithm is k-competitive, any marking algorithm must also be k-competitive.